Improving System Performance in Case-Based Iterative Optimization through Knowledge Filtering

Kazuo Miyashita *
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba
Ibaraki 305, JAPAN
miyasita@etl.go.jp

Katia Sycara
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, U.S.A.
katia@cs.cmu.edu

Abstract

Adding knowledge to a knowledge-based system is not monotonically beneficial. We discuss and experimentally validate this observation in the context of CABINS, a system that learns control knowledge for iterative repair in ill-structured optimization problems. In CABINS, situation-dependent user's decisions that guide the repair process are captured in cases together with contextual problem information. During iterative revision in CABINS, cases are exploited for both selection of repair actions and evaluation of repair results. In this paper, we experimentally demonstrated that unfiltered learned knowledge can degrade problem solving performance. We developed and experimentally evaluated the effectiveness of a set of knowledge filtering strategies that are designed to increase problem solving efficiency of the intractable iterative optimization process without sacrificing solution quality. These knowledge filtering strategies utilize progressive case base retrievals and failure information to (1) validate the effectiveness of selected repair actions and (2) give-up further repair if the likelihood of success is low. The filtering strategies were experimentally evaluated in the context of job-shop scheduling, a well known ill-structured problem.

1 Introduction

Recently, there has been increased interest in the issue of the *utility of knowledge* in knowledge-based systems. Several studies [Markovitch and Scott, 1993; Minton, 1988] have defied the traditional belief that increasing a problem solver's knowledge is a monotonically beneficial process. Utility of knowledge depends on the difference between its cost (e.g. cost of knowledge acquisition, storage, matching) and its benefits (e.g. increased problem solving efficiency, increased solution quality). Operational definitions of utility of a knowledge item in the literature (e.g. [Minton, 1988]) state that it is the difference between the cost of solving a problem with the knowledge item and solving it without it. Machine learning systems can acquire greater amounts of knowledge than is possible to be hand-coded for them by their developers. Because less intelligence is involved in the automated knowledge acquisition by machine learning systems, the acquired knowledge may be of low quality (e.g. it could be incorrect, irrelevant or redundant). Hence it is very important for machine learning systems to consider the quality of the knowledge they employ and develop heuristic strategies to eliminate harmful knowledge, i.e. knowledge whose elimination from consideration would improve problem solving performance. [Markovitch and Scott, 1993] has termed such strategies *information filters* because they filter out harmful knowledge from being used in problem solving. Examples of information filters include discarding of least useful board positions in the Checkers Player [Samuel, 1959], selecting only misclassified instances in ID3 [Quinlan, 1986], and estimating the utility of newly acquired control rules and deleting those unlikely to be useful in PRODIGY [Minton, 1988]

To ascertain the utility of information filtering, it has to be shown that learned unfiltered knowledge is harmful in the sense that its addition to a system's knowledge base deteriorates system performance along some evaluation criterion. In this paper, we experimentally show non-beneficial effects of unfiltered learned knowledge in a system called CABINS and study the effectiveness of three filtering strategies designed to fix the problem. CABINS is a case-based system that learns *control knowledge* through user interaction and utilizes it for solution improvement through iterative repair in job shop scheduling, an ill-structured domain [Miyashita and Sycara, 1995]. The information filtering idea can be applied in a CBR system in various ways. One way could be to eliminate cases from the case base ("forgetting" [Kolodner, 1993]). Another way would be to prune out parts of the case base from consideration during problem solving. Traditionally, this is done in CBR systems by using some sort of similarity metric. By adjusting the similarity metric a larger or smaller number of cases are allowed to pass this filter and be considered during problem solving. Alternatively, instead of adjusting the similarity metric, various filtering tests could be applied to selectively narrow down retrieved knowledge from the cases. A fourth method could be filtering out

*Previously at Matsushita Electric Industrial Co. Ltd.

information retrieved from cases, for example filtering out failure information. Yet a fifth method could be to use retrieved knowledge to extract new indices for subsequent retrieval thus using successive retrievals as filters. The filtering strategies we have developed involve progressive information filtering using a combination of successive retrievals and tests and utilizing success and failure information stored in cases.

The use of failure information in CBR systems is not new. For example, CHEF [Hammond, 1989] assumes the existence of a model-based simulator for evaluating a derived plan and detecting failure and uses handcrafted domain rules for selecting repair actions. Work by [Kambhampati and Hendler, 1992] aims to speedup system performance through learning and case-based reuse of control rules in planning, where a correct satisficing plan is sought and a strong domain model is assumed. The contribution of our work is to experimentally study failure-based information filtering strategies in case-based control knowledge learning for an *optimization task* in a domain *without a strong domain model.* CABINS was evaluated in the task of schedule optimization in the job shop scheduling domain. Our experimental results, presented in section 5.4 show that progressive filtering is superior to flat filtering in that it enables CABINS to increase both solution quality and search efficiency. In particular, CABINS uses its past failure experiences in progressive filtering to (1) *validate* use of a selected repair action, and (2) *interrupt* a repair that is unlikely to produce useful results. Given our encouraging experimental results, we believe that our filtering strategies can be used as a domain-independent refinement of k-Nearest Neighborhood retrieval for control knowledge learning tasks.

2 The Task Domain

Scheduling assigns a set of tasks over time to a set of resources with finite capacity. One of the most difficult scheduling problem classes is job shop scheduling. Job shop scheduling is a well-known NP-complete problem [French, 1982]. In job shop scheduling, each task (heretofore called a job or an order) consists of a set of activities to be scheduled according to a partial activity ordering. The dominant constraints in job shop scheduling are: temporal precedence constraints that specify the relative sequencing of activities within a job and resource capacity constraints that restrict the number of activities that can be assigned to a resource during overlapping time intervals. The activity precedence constraints along with a job's release date and due date restrict the set of acceptable start times for each activity. The capacity constraints restrict the number of activities that can use a resource at any particular point in time and create conflicts among activities that are competing for the use of the same resource at overlapping time intervals. The goal of a scheduling system is to produce schedules that respect temporal relations and resource capacity constraints, and optimize a set of objectives, such as minimize tardiness, maximize resource utilization etc.

The tight interactions between temporal precedence and capacity constraints give rise to many nonlinear effects during search for an optimal schedule. It is in general impossible to have a strong domain model that could help estimate the effects of system decisions on optimization objectives. The use of heuristics and study of experimental results is the typically used method for job shop schedule optimization [French, 1982].

3 CABINS Overview

In order to make the information filtering aspect comprehensible, we give here a brief and of necessity incomplete overview of CABINS. For more details on CABINS implementation, operation and other experimental results that describe different CABINS capabilities, the reader is referred to [Miyashita and Sycara, 1995].

CABINS *incrementally revises a complete but suboptimal schedule* to improve its quality. Revision consists in identifying and moving activities in the schedule. To move activities, CABINS uses a number of repair operators, called repair tactics that it possesses. In the current implementation, CABINS has 11 repair tactics (e.g., jump_left, which moves an activity as much to the left on the timeline as possible without violating precedence constraints). Because of the tight constraint interactions, a revision in one part of the schedule may cause constraint violations in other parts. It is generally impossible to predict in advance either the extent of the constraint violations resulting from a repair action, or the nature of the conflicts because of interacting influences of constraint propagations. Therefore, a repair operator must be selected, applied and its repair outcome must be evaluated in terms of the resulting effects on scheduling objectives.

Thus, at each decision point, CABINS must make three control decisions: which activity to move, which repair operator to apply and how to evaluate the repair result so as to eventually optimize overall schedule quality according to scheduling objectives. Each of these decisions is very difficult for the following reasons. A schedule typically contains many jobs, each of which contains many activities. There is no a priori knowledge as to the order of activity moves that will give the best result. Moreover, we have found no distinguishing features of an activity per se that would allow satisfactory matching with previous similar activities. Therefore, CABINS orders the jobs randomly and within each job repairs activities from the first to the last [Miyashita and Sycara, 1995]. Since activities get moved during repair, an activity may be repaired (moved) multiple times during a problem solving session. Selecting a repair operator to apply is also very difficult. Unlike traditional planning operators, the repair operators in CABINS have no well defined preconditions or postconditions. Hence learning operator selection is fraught with all the difficulties of learning control rules in traditional planning plus the additional complexity of not knowing the operator pre- and post-conditions. CBR enables CABINS to learn a control model of repair operator selection. Finally, evaluation of the result of applying a repair operator is also very difficult, since the optimization objectives are often context and user dependent and reflect state-dependent tradeoffs that are difficult to describe in a simple man-

ner. CABINS learns through CBR what combinations of effects of application of a particular repair action constitutes an acceptable or unacceptable repair outcome.

In each repair iteration, CABINS focuses on one activity at a time, the *focal_activity*, and tries to repair it. *A case in CABINS describes the application of a particular modification to a focal_activity.* A case in CABINS comprises 3 types of the features: global features, local features and repair history. The global features reflect an abstract characterization of potential repair flexibility for the whole schedule. Associated with the focal_activity are local features that we have identified and which potentially are predictive of the effectiveness of applying a particular repair tactic. Because of the ill-structuredness of job shop scheduling, local and global features are heuristic approximations that reflect problem space characteristics.

The repair history records the sequence of applications of successive repair actions, the repair outcome and the effects. Repair effect values describe the impact of the application of a repair action on scheduling objectives. A repair outcome is the evaluation assigned to the set of effects of a repair action and takes values in the set ['acceptable', 'unacceptable']. Typically the outcome reflects tradeoffs among different objectives. An outcome is 'acceptable' (i.e. a success) if the user accepts the tradeoffs involved in the set of effects for the current application of a repair action. Otherwise, it is 'unacceptable', a failure. The effect salience is assigned by the user when the outcome is 'unacceptable', and it indicates the significance of the effect to the repair outcome. The user's judgment as to balancing favorable and unfavorable effects related to a particular objective constitute the explanations of the repair outcome.

Once enough cases have been gathered through user interactive schedule repair, CABINS repairs sub-optimal schedules without further user interaction. CABINS repairs the schedules by (1) sorting jobs in a random order, (2) focusing on a focal_activity to be repaired in each repair cycle, (3) invoking CBR with global and local features as indices to decide the most appropriate repair tactic to be used for each focal_activity, (4) invoking CBR using the repair effect features (type, value and salience) as indices to evaluate the repair result, and (5) when the repair result is deemed a failure, deciding whether to give up or which repair tactic to use next. The similarity metric that CABINS uses is a k-Nearest Neighborhood method.

4 Experimental Design

To test whether learned unfiltered knowledge is beneficial or not, we used controlled experimentation on a benchmark suite of job shop scheduling problems. These problems are variations of the problems we have used to test other aspects of CABINS [Miyashita and Sycara, 1995]. The benchmark problems have the following structure: each problem has 10 orders of 5 activities each. Six groups of 10 problems each were randomly generated by considering three different values of the range parameter, and two values of the bottleneck configuration.

A cross-validation evaluation method was used. Each problem set of each problem class was divided into two groups with the same size (i.e., each group contains 30 problems). Currently we have collected about 12000 cases for each group of training sets. These cases were then used for case-based repair of the test problems in the other group. The above process was repeated by interchanging the training sets.

In the experiments reported here, we used the sum of tardiness and inventory costs as an explicit cost function to reflect optimization criteria[1]. We built a greedy hill-climb rule-based optimizer (RBR) that goes through a trial-and-error repair process to optimize a schedule by minimizing the cost function. RBR was implemented to try what seems a best repair action first based on heuristic evaluation of its possible repair effects and go through all the available repair actions before giving up further repair. For each repair, the repair effects were calculated and, on this basis, since RBR had a pre-defined cost function, it could evaluate repair results precisely. RBR was used as the experimental baseline where no learning of control knowledge takes place. RBR was also used as a case base builder for CABINS. Each time RBR repaired an activity, the repair decisions made by RBR such as repair action selection and repair result evaluation were recorded in a case. In this set of experiments, RBR was used to create a case base with 12000 cases. Naturally, a cost function, though known to RBR, is not known to CABINS and is only implicitly and indirectly reflected in an extensional way in the case base.

In our experiments, the number of repair tactic applications was used as a metric of search efficiency for a number of reasons: (1) tactic application involves constraint propagation and calculation of repair effects, thus being the most computationally expensive process in schedule repair, (2) the number of tactic applications is a machine independent measure, as opposed to for example, CPU time, and (3) the number of tactic applications directly reflects the number of search states examined. Reduction of the number of tactic applications improves the efficiency of iterative schedule revision process.

Table 1: Performance in unfiltered learning of control knowledge

	Solution Cost	Tactic Applications
Unfiltered	698.8	2206.2
RBR	752.4	1229.8

In Table 1 "Unfiltered" indicates the performance of CABINS, which retrieves similar successful repair experiences and applies the repair tactic that was dominant in the most similar success cases. If the tactic is evaluated as unacceptable, in the Unfiltered condition, CABINS automatically tries another repair tactic used in the next most similar cases.

Table 1 shows that unfiltered learned knowledge is indeed harmful in terms of problem solving efficiency al-

[1]Such a cost function is widely accepted in practical scheduling environments.

though it produces solutions of similar quality as compared to the base condition of RBR. This result means that the cost of utilizing the learned knowledge is excessive. We think that this is due to the Unfiltered repair's *greediness* and *stubbornness*. Greediness makes Unfiltered repair applies the tactic from the most similar success case no matter how small the similarity. Stubbornness makes the Unfiltered repair insists on trying to repair the current focal_activity without giving up if the likelihood of a successful repair seems small.

5 Information Filtering

To decrease the number of tactic applications without sacrificing solution quality, filtering strategies must be found that retrieve more relevant tactics and better estimate the likelihood of successful outcome before application of a selected tactic. We hypothesized that incorporating failure information in filters would alleviate non-beneficial effects of potentially harmful learned knowledge by lowering the cost of excessive number of tactic applications. So we focused on designing filters that would enable CABINS to (1) learn to avoid repeating similar previous failures, and (2) learn to avoid wasting lots of efforts in trying to solve too difficult problems.

To analyze the correctness of the above hypothesis, the following three filtering strategies were experimentally implemented in CABINS: *validated filtering*, *interruptive filtering* and *hybrid filtering*. These filtering methods were tested using the same problems, cost function and case bases used for the experiment in Section 4.

5.1 Validated Filtering

The validated filtering method allows CABINS to apply a repair action only after it has been validated as possibly effective for repairing the current problem. Validation techniques have been used in case-based reasoning systems (e.g. [Hammond, 1989; Simoudis and Miller, 1991]). For example, in CASCADE and COAST [Simoudis and Miller, 1991], a validation process extracts from the case base only those cases that appear to be closely relevant to the current problem using a *validation model* of the domain which consists of knowledge of interrelations among diagnostic tests and knowledge about individual diagnostic tests and their expected values in previous cases that would match a current problem.

The above systems utilized established models based on deep understanding of their domain for validation. Domain models act as filters that allow only validated information to be used in problem solving. However, in ill-structured domains like job shop scheduling, neither causal model nor validation model of the domain are readily available. In the job shop scheduling problem, non-linearity of objectives and tight interactions of constraints make it hard to predict the effects of a local optimization decision on global optimality, thus making the analysis of the domain structure required to build a causal model and a validation model extremely difficult.Therefore, CABINS must validate the potential effectiveness of a retrieved repair tactic without such models. To compensate for lack of these models, CABINS utilizes failure information as a filter through which only validated repair tactics can pass.

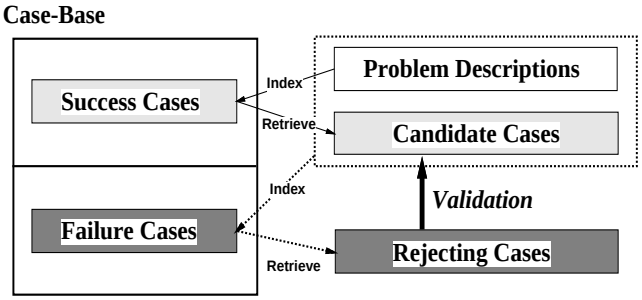

Figure 1: Validation filtering process in CABINS

Fig. 1 shows the schematic diagram of the validation filtering process in CABINS. First, using global and local features as indices, candidate cases are retrieved from *success cases* that store successful repair episodes in a case base. Then, another CBR retrieval is done from *failure cases* using as additional indices the repair tactics indicated in the already retrieved successful cases. This retrieval results in a set of *rejecting cases*. The candidate repair tactics found in rejecting cases with low credibility (i.e. low similarity to the current problem), are then allowed to be used since they have been validated to be possibly effective (i.e. not likely to fail).

The repair tactic selection procedure using validated filtering is as follows:

1. Select k [2] nearest cases of the current problem that succeeded in repair using global and local case features as indices (these are labelled "candidate cases" in Fig. 1).
2. Among the candidate cases, obtain the sum of case similarity [3] to the current problem for each of the repair tactics which was successfully appiled in the cases.
3. Sort the repair tactics of step 2 in decreasing order of the corresponding value of the similarity sums.
4. Until no repair tactic remains in the sorted queue, do the followings:
   (a) Pick up the next repair tactic in the sorted repair tactics queue.
   (b) Using as indices, global and local features and the picked-up repair tactic, select k nearest cases of the current problem, in which an application of the picked-up tactic resulted in failure ("rejecting cases" in Fig. 1).
   (c) Sum up the similarity of the selected k "rejecting cases".
   (d) If the sum is smaller than a pre-defined *threshold value*, return the tactic as a validated tactic.
5. Return "give_up" as a repair tactic, i.e. give up trying to repair the current activity.

[2] In our experiments, we heuristically decided k as 5.

[3] For details of the similarity calculation, see [Sycara and Miyashita, 1994].

The validated filtering procedure allows only validated tactics to be considered for repair application. CABINS judges the likelihood that a selected repair tactic will fail in the current problem from the past failure experiences of the same tactic. If CABINS judges that a tactic is not likely to fail in the current problem (i.e. the sum of similarity of rejecting cases associated with the tactic is less than some threshold), CABINS considers the repair tactic as validated.

Since in difficult problems such as schedule repair, failures usually outnumber successes, if the value of the threshold (in step 4(d) of the above procedure) is defined as moderately high, overly pessimistic results could be produced (i.e. CABINS seldom validates a tactic). To avoid this, a threshold value was set as 4.99. Since the value of k was 5 in the experiments, validation rejection was less likely to occur.

5.2 Interruptive Filtering

In validated filtering, CABINS gives up repairing a particular activity either when there are no validated tactics, or when all validated tactics have resulted in failure (i.e. an unacceptable repair outcome). The interruptive filtering method allows CABINS to shift its repair attention to another activity when repair of the current activity is estimated too difficult. CABINS gives up a further repair when it determines that it would be a waste of time. If there are enough failure cases in the past similar situations to the current one, CABINS gives up repairing the current activity, and switches its attention to another.

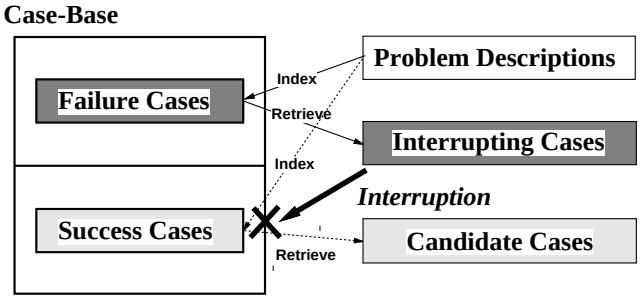

Figure 2: Interruptive filtering process in CABINS

Fig. 2 shows the schematic diagram of the interruptive filtering process in CABINS. In the process, cases that match the current problem are retrieved from *failure cases* (the retrieved cases are called *interrupting cases* in Fig. 2). If the interrupting cases have high credibility (i.e. high similarity to the current problem), the process of retrieving candidate cases to use in repairing the current focal_activity is not allowed to proceed and the problem solver seeks another activity to repair.

The repair tactic selection procedure using interruptive filtering in CABINS is as follows:

1. Using global and local features as indices, retrieve k nearest cases of the current problem from failure cases (i.e., "interrupting cases" in Fig. 2).
2. Sum up the similarity of the selected k "interrupting cases".
3. If the sum exceeds a pre-defined *threshold value*, return "give_up" as a repair tactic, i.e. give up repairing the current focal_activity.

   Otherwise, select k nearest cases of the current problem using global and local features as indices from successful cases (these are the "candidate cases" in Fig. 2 and they are the same as the "candidate cases" in Fig. 1 for a given focal_activity).
4. Among the candidate cases, obtain the sum of case similarity to the current problem for each of the repair tactics which was successfully appiled in the cases.
5. Sort the repair tactics of the previous step in decreasing order of the corresponding value of the similarity sums.
6. Return the first repair tactic in the sorted repair tactic queue as the selected repair tactic.

In the above procedure, CABINS judges the likelihood that a current problem cannot be repaired from failure experiences in past similar problems. If CABINS judges that the current problem is not likely to be repaired (i.e. the number of failures of similar past problems is high enough), CABINS gives up repairing the current problem. If CABINS decides that the current problem is not too difficult, it proceeds to repair it selecting repair tactics from success cases without tactic validation. As was discussed in the validated filtering method, in difficult problems such as schedule repair, failures usually outnumber successes and if the value of the threshold (in step 3 of the above procedure) is defined moderately high, overly pessimistic results could be produced (i.e. CABINS suggests giving up too often). To avoid this, a value of the threshold was set as 4.99.

The ways failure cases are used in validated and interruptive filtering seem to be independent and compatible with each other. Each filtering strategy filters different knowledge at different stages of repair action retrieval and evaluation. Hence, the two strategies can be combined in a serial fashion to form a hybrid filtering strategy.

5.3 Hybrid Filtering

In hybrid filtering, CABINS first decides whether to give up repairing the current focal_activity using "interrupting cases" (as in Interruptive Filtering). If it is decided that repair of the current focal_activity should proceed, CABINS uses Validated filtering to validate the selected tactics that will be tried in the repair.

The repair tactic selection procedure using hybrid filtering in CABINS is as follows:

1. Using global and local features as indices, retrieve k nearest cases of the current problem from failure cases (i.e. "interrupting cases" in Fig. 2).
2. Sum up the similarity of the selected k "interrupting cases".
3. If the sum exceeds a pre-defined *threshold value*, return "give_up" as a repair tactic, i.e. give up repairing the current focal_activity.

Otherwise, use the validated filtering procedure (Fig. 1).

5.4 Experimental Results

Table 2: Performance comparison among U(nfiltered), R(BR), V(alidated), I(nterruptive) and H(ybrid)

	Solution Cost	Tactic Applications	CPU (sec)
U	698.8	2206.2	551.1
R	752.4	1229.8	N/A
V	751.1	920.3	435.9
I	724.3	1186.8	528.1
H	753.4	776.6	424.1

Table 2 shows performance comparison among the unfiltered condition, RBR, and the three filtering strategies. The results in the table show that CABINS with validated filtering improved its efficiency about 58% as compared with the unfiltered condition without unduly sacrificing schedule quality. CABINS with validated filtering was even faster than RBR. Table 2 also shows that CABINS with interruptive filtering improved its efficiency about 46% as compared with unfiltered condition while maintaining high schedule quality.

From the results in the table, it is also shown that CABINS with the hybrid repair improved its efficiency about 65% compared with the unfiltered condition without reducing schedule quality. The results also suggest that CABINS with hybrid filtering produced high quality schedules more efficiently than RBR.

To analyze the effects of combining two repair methods in the hybrid repair, the number of tactic application using validated filtering and using interruptive filtering were compared with that of hybrid filtering. By hypothesis testing, it was found that CABINS with hybrid filtering should be more efficient than CABINS with interruptive repair or CABINS with validated repair. Therefore, by combining the validated repair and the interruptive repair, a synergistic effect emerges in improving the efficiency of the repair process.

Although experimental results show that the three information filtering strategies were effective in reducing the number of tactic applications in CABINS, it could be argued that knowledge filtering will also consume computational resources. CPU time is a generally accepted metric of overall problem solving cost. The averaged CPU time results in Table 2 show a pattern similar to the one observed for number of tactic applications (i.e., validated repair, interruptive repair and hybrid repair consume less CPU time than unfiltered repair) in our experiments. These results are encouraging.

6 Conclusions

We discussed the phenomenon of non-beneficial effects, in terms of degradation of the performance of a problem solver, of learned knowledge. We showed the existence of this phenomenon in the context of case-based learning of control knowledge. This knowledge is used to to guide revision-based optimization in an ill-structured domain, job shop scheduling. To alleviate the non-beneficial effects of learned knowledge, we designed and experimentally demonstrated the effectiveness of three knowledge filtering strategies. These strategies, validated filtering, interruptive filtering and hybrid filtering employ a combination of successive case retrievals and in particular exploit failure information found in cases in order to (1) validate learned control actions, or (2) avoid wasting effort in attempting repairs where the likelihood of success is low. Our experimental results show that the knowledge filtering strategies were effective in increasing the efficiency of problem solving without compromising solution quality. We are currently exploring automated adaptation of the filtering control parameters (value of k and of the threshold).

References

[French, 1982] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, London, 1982.

[Hammond, 1989] Kristian J. Hammond. *Case-Based Planning : Viewing Planning as a Memory Task*. Academic Press, New York, NY, 1989.

[Kambhampati and Hendler, 1992] Subbarao Kambhampati and James A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, 1992.

[Kolodner, 1993] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA, 1993.

[Markovitch and Scott, 1993] Shaul Markovitch and Paul D. Scott. Information filtering: Selection mechanisms in learning systems. *Machine Learning*, 10:113–151, 1993.

[Minton, 1988] S. Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer Academic Publishers, Boston, MA, 1988.

[Miyashita and Sycara, 1995] Kazuo Miyashita and Katia Sycara. CABINS: A framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence*, 1995. To appear.

[Quinlan, 1986] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Samuel, 1959] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3:211–229, 1959.

[Simoudis and Miller, 1991] E. Simoudis and J.S. Miller. The application of CBR to help desk applications. In *Proceedings of the Case-Based Reasoning Workshop*, pages 25–36. DARPA, 1991.

[Sycara and Miyashita, 1994] Katia Sycara and Kazuo Miyashita. Case-based acquisition of user preferences for solution improvement in ill-structured domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 44–49, Seattle, WA, 1994. AAAI.